CSO-Recitation 05

CSCI-UA 0201-007

R05: Assessment 03 & Pointers & Arrays

Today’s Topics

- Assessment 04
- Pointers
- Arrays

Assessment 04

Q1 Pointers and arrays

Given variable definition char *c[10]; what is the type of the expression c[0]+1?

A. char **

[✓ B. char *]

C. char

D. none of the above

char *c[10]:

- c is an array of pointer to char
  - type of c: char **
- c[0] ==*c
  - type of c[0]: char *
- ~~c[0]+1 == *(c+1) ?~~
- c[1] ==*(c+1)
- c[0]+1 == *c+1
  - also pointer arithmetic
  - type of c[0]+1: char *

e.g: [“cso”, “recitation”, …, “TA”]

- c[0]+1 is char*, then what is the value of *(c[0] + 1)?

c[0]: the pointer to “cso”

c[0] + 1: points to?

*(c[0] + 1) = ‘s’

Q2 Pointers and arrays

Given variable definition char *c[10]; what is the type of the expression c+1?

[✓ A. char **]

B. char *

C. char

D. none of the above

c+1 == &c[1]

Q3 Pointers and arrays

- e.g. c=[’c’, ‘s’, ‘o’, …’r’]
- c[0]==‘c’
- c[1]==‘s’
- c[0]+1==‘d’

Given variable definition char c[10]; what is the type of the expression c[0]+1?

A. char **

B. char *

[✓ C. char]

D. none of the above

char c[10]:

- c is an array of char
  - type of c: char *
- c[0] ==*c
  - type of c[0]: char
- c[0]+1 == *c+1
  - type of c[0]+1: char

Q4 Pointers and arrays

Given variable definition char c[10]; what is the type of the expression c+1?

A. char **

[✓ B. char *]

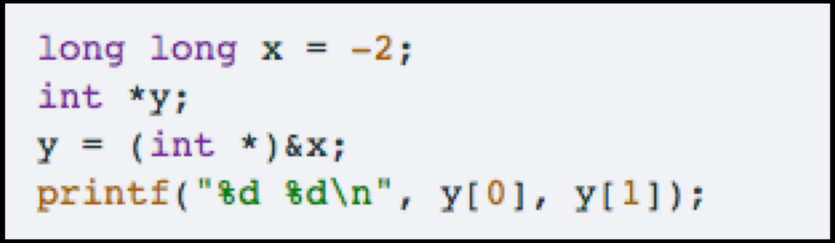
C. char

D. none of the above

c+1 == &c[1]

Q5 Pointer casting

What's the output of the following code fragment (assuming it runs on a 64-bit little endian machine):



```
long long x = -2;
int *y;
y = (int *)&x;
printf("%d %d\n", y[0], y[1]);
```

A. -1 -1

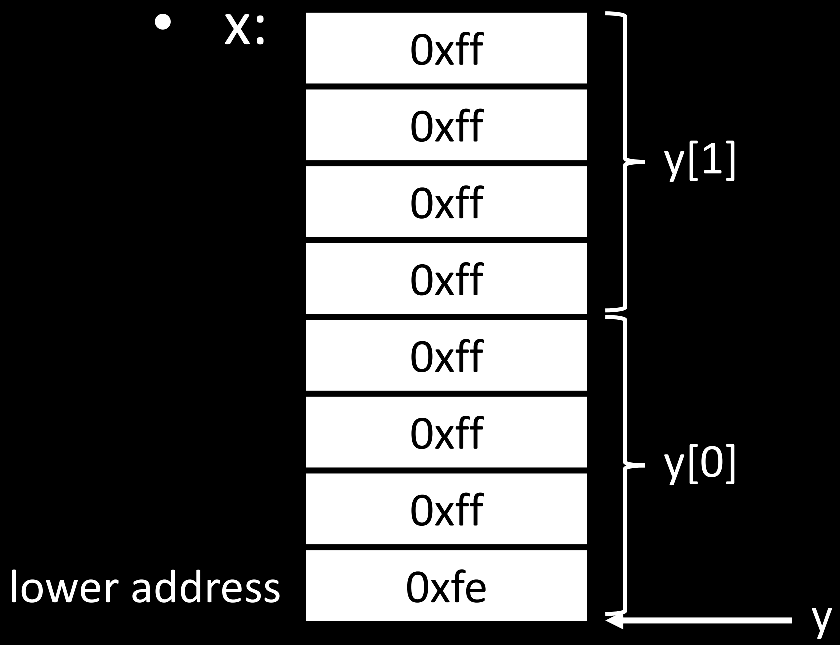
B. -2 -2

C. -1 -2

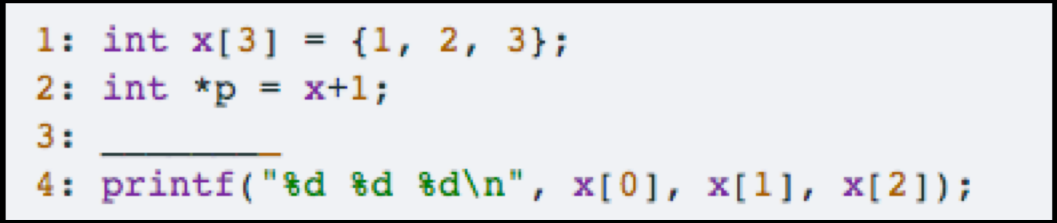
[✓ D. -2 -1]

E. Segmentation fault

F. None of the above



Q6 Pointer arithmetic



```
1: int x[3] = {1, 2, 3};
2: int *p = x+1;
3: ________
4: printf("%d %d %d\n", x[0], x[1], x[2]);
```

Here's a C code fragment. In order for the above code fragment to output 1 2 10, which of 1 line of code that you should put at Line-3?

A. p[0] = 10;

[✓ B. p[1] = 10;]

C. p[2] = 10;

D. *(p) = 10;

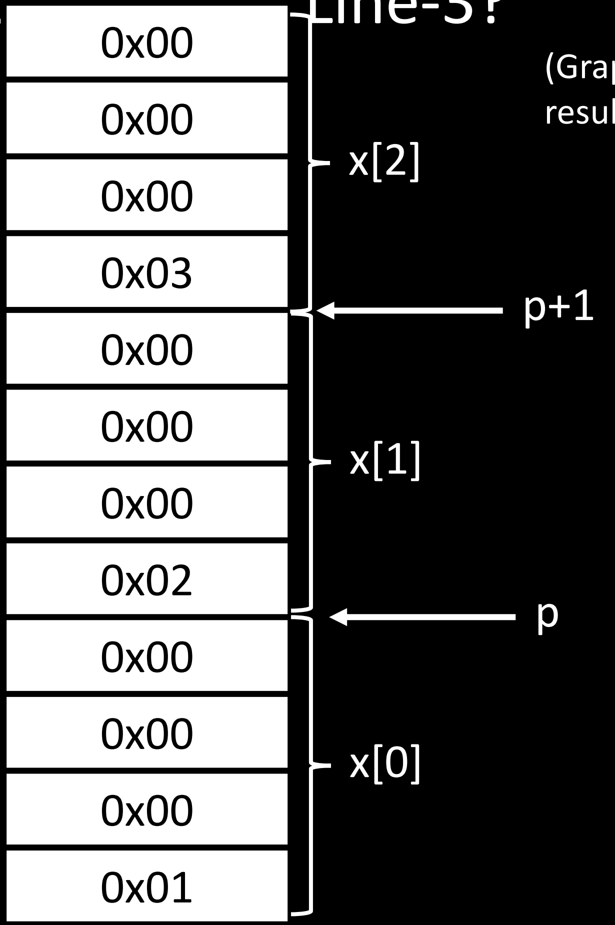
[✓ E. *(p+1) = 10;]

F. *(p+2) = 10;

G. p++;

H. p--;

- x:



0x00
0x00
0x00
0x03
0x00
0x00
0x00
0x02
0x00
0x00
0x00
0x01

x[2]
p+1
x[1]
p
x[0]

(Graph drawn assuming little endian, but the result is the same for large endian too)

Q7 ASCII

Suppose char c stores some ASCII character. What could be its value interpreted as a signed 1-byte integer?

A. any integer in the range [-128,127]

B. any integer in the range [0, 255]

[✓ C. any integer in the range [0, 127]]

D. any integer in the range [-1, 255]

- ASCII characters:
- use one byte (with MSB=0) to represent each character

- if it is interpreted as a signed 1-byte int:
  - smallest: 00000000 -> 0
  - largest: 011111111 -> 127

Q8 String

```
1: char c = 'a';
2: int x = strlen(&c);
```

What's the value of x after the above two lines of code?

A. Compilation error at line 1

B. Compilation error at line 2

C. x = 0

D. x = 1

E. x = 2

F. x = 3

[✓ G. x's value is undefined (i.e. could be any int value).]

- What is C's solution to determine string length?
  - Programmers are expected to store a NULL character at the end of the string (by convention)
  - Count the #char until '\0'

Q9 String

```
1: char c = ‘\0';
2: int x = strlen(&c);
```

What's the value of x after the above two lines of code?

A. Compilation error at line 1

B. Compilation error at line 2

[✓ C. x = 0]

D. x = 1

E. x = 2

F. x = 3

G. x's value is undefined (i.e. could be any int value).

- What is C’s solution to determine string length?
  - Programmers are expected to store a NULL character at the end of the string (by convention)
  - Count the #char until ‘\0’

Q10 String

Exercise:

- what if `int x = strlen(&a);` ?
- what if ` int a = 0x01414243;`?

```
1: int a = 0x00414243;
2: int x = strlen((char *)&a);
```

What's the value of x after the above two lines of code?

A. Compilation error at line 1

B. Compilation error at line 2

C. x = 0

D. x = 1

E. x = 2

[✓ F. x = 3]

G. x's value is undefined (i.e. could be any int value).

- What is C's solution to determine string length?
  - Programmers are expected to store a NULL character at the end of the string (by convention)
  - Count the #char until '\0'
- (char *)&a -> casting to char *

Pointers

A variable that stores a memory address

What are pointers?

- They are variables that store addresses
  - Pointers can have different types, depending on what they point to
    - But they remain the same size – for us on a 64-bit system, 8 bytes (64 bits)

Type	Value	Address
int	an integer number	memory address
float	a floating point number	memory address
char	a character/byte	memory address
pointer	memory address	memory address

- If I want the value of a variable var -> var
- If I want the address of a variable var -> &var
- If var is a pointer, then I can get the value of the variable that var points to -> *var

What are pointers?

- They are variables that store addresses
  - Pointers can have different types, depending on what they point to
    - But they remain the same size – for us on a 64-bit system, 8 bytes (64 bits)
- Two primary operations
  - & - called “reference”
    - Gets the address of a variable / array element
    - You perform this to get the value for a pointer
  - * - called “de-reference”
    - Gets the value located at a memory address
    - You perform this on the pointer

How do you use pointers?

- Say you have a variable var
  - int var = 10;
- You can make a pointer called ptr using this code
  - int *ptr;
- ptr can be set to point to var with the reference operator
  - ptr = &var;
- The value of ptr is now the address of var, not its value
  - To get the value, de-reference:
    - *ptr //this equals to 10
    - *ptr = 5; // this sets var to 5

Pointer types

- Why do we need pointer types?
  - Without it, making mistakes like de-referencing a number by accident would be common
  - Without it, pointer arithmetic wouldn’t work
- What is pointer arithmetic?
  - If you have a pointer called ptr, the value of ptr+1 is based on the type of ptr
    - If ptr is a char*, then ptr+1 is the memory address of next char after ptr
    - If ptr is an int*, then ptr+1 is the memory address of next int after ptr
  - ptr+n means “start at ptr, and go forward as many bytes as *n* copies of what ptr points to take up”

Function arguments and pointers

- In C, arguments are passed by value
  - Means that when you call a function, the arguments are copied from the caller to the function's stack frame
  - This means that if a function modifies one of its arguments, it is not modified for whoever called the function
- If you want to pass a reference, you must use pointers
  - Then the function can modify the variable by dereferencing the pointer

Arrays

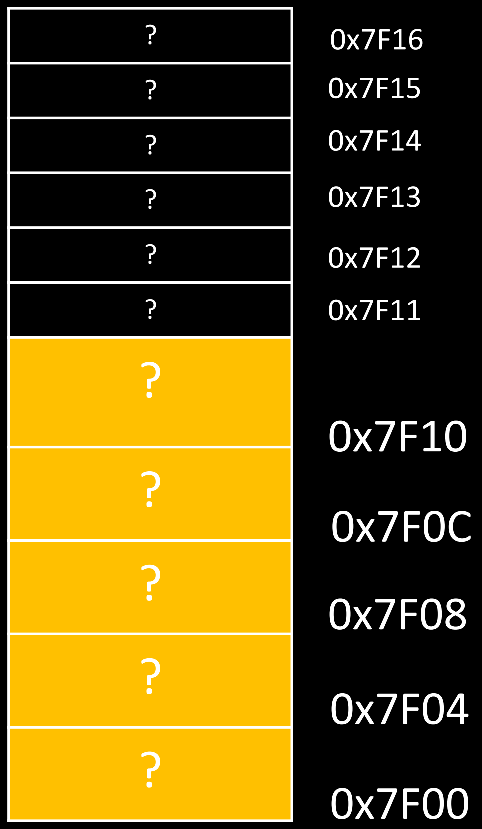
Contiguous, homogenous data

What are arrays?

- Basically, they are chunks of memory that hold a number of elements of the same data type
- This memory is contiguous, that is, the elements are all touching
- You can define an int array like this
  - int my_array[5];
  - This will make an array of 5 ints (20 bytes)
  - You can initialize the array as follows:
    - int my_array[5] = {1, 2, 3, 4, 5};
    - You can also set it to all zeroes using int my_array[5]={0};
- You can index with the [] operator
  - my_array[0] gets the first element of my_array
  - my_array[0] = 5 sets the first elelment of my_array to 5

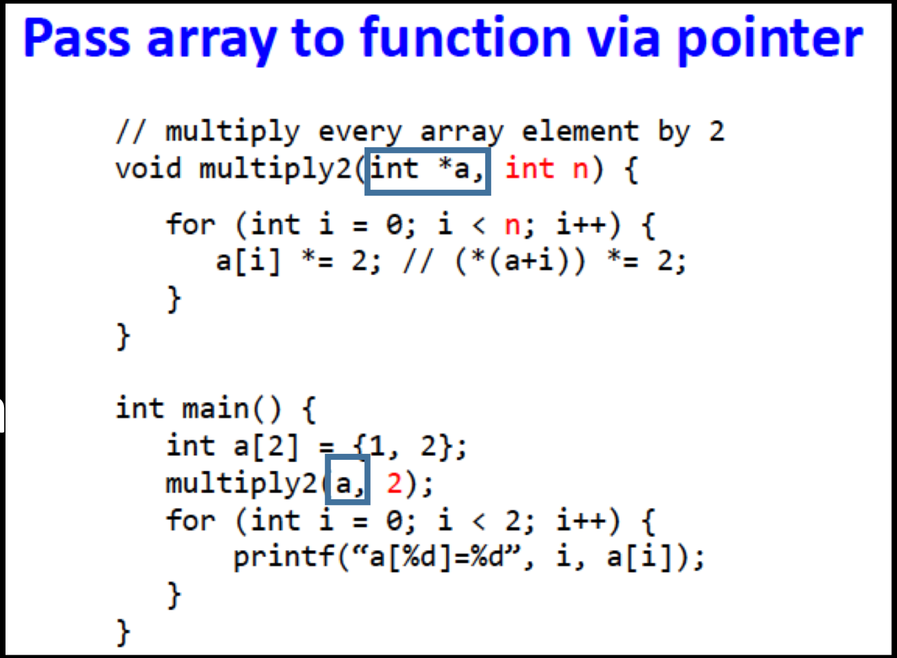
Defining an array

- int arr[5];
- The value of an array is the address of its first element
  - The value of arr is 0x7F00
    - arr==&arr[0]
- Let a pointer points to the 1st element of this array
  - int *p = arr;
    - int *p = &arr[0];
- Array and pointer can be syntactically equivalent
  - *p == p[0]==arr[0]
  - *arr ==arr[0]
  - *(arr+2) ==arr[2]



?	0x7F16
?	0x7F15
?	0x7F14
?	0x7F13
?	0x7F12
?	0x7F11
?	0x7F10
?	0x7F0C
?	0x7F08
?	0x7F04
?	0x7F00

Pointer and array



Pass array to function via pointer

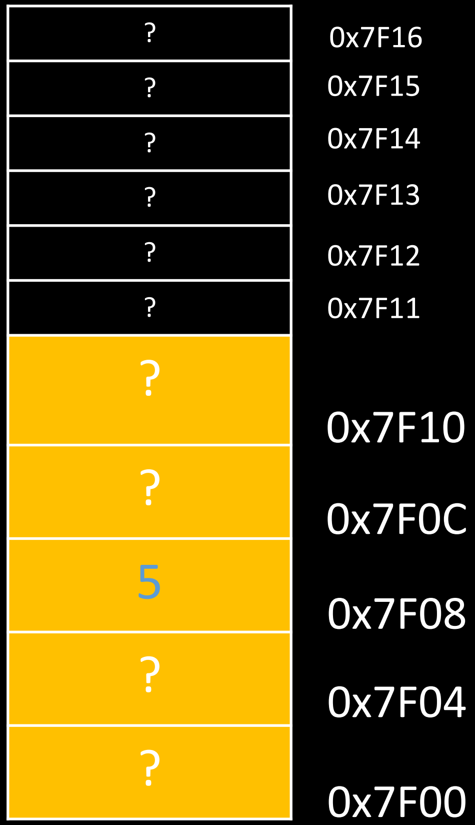
```
// multiply every array element by 2
void multiply2(int *a, int n) {
   for (int i = 0; i < n; i++) {
      a[i] *= 2; // (*(a+i)) *= 2;
   }
}

int main() {
   int a[2] = {1, 2};
   multiply2(a, 2);
   for (int i = 0; i < 2; i++) {
       printf(“a[%d]=%d”, i, a[i]);
   }
}
```

- One difference between an array name and a
  - A pointer is a variable
    - p = arr; / p++; are legal
  - But an array name is not a variable..
    - cannot write things like arr++; / arr=p; (illegal)
- When an array name is passed to a function,
  - What it passed is the address of the 1st element
  - Oftentimes we use a pointer type to accept it
    - Within the called function, this argument is a local variable, and an array name parameter is a pointer, that is, a variable containing an address
  - But we need to also pass the number of elements in this array to function

Indexing an array

- int arr[5];
- Arrays can be index like so
  - arr[2] = 5;
  - This will set the third element of arr to 5
  - This is the same as *(arr + 2) = 5;
    - Which is to say, this is done by taking the value of arr, 0x7F00, and adding 2 to it according to pointer arithmetic
    - The size of int is 4, so we are going 8 bytes passed arr, 8 + 0x7F00 = 0x7F08



? 0x7F16

? 0x7F15

? 0x7F14

? 0x7F13

? 0x7F12

? 0x7F11

? 0x7F10

? 0x7F0C

5 0x7F08

? 0x7F04

? 0x7F00

Pointers to pointers (Pointer arrays)

- Since pointers are variable themselves, they can be stored in arrays just as other variables can
  - char *a[2];
- Let a pointer points to the 1st element of this array (of pointers)
  - char **p = &a[0]; / char **p=a;
- An array of pointers
- Think about what can this do?